



Git

Denise Nicole Krzyny

Software Developer

denisek@lagash.com

Agenda

01

Breve historia

Control de versiones, cómo y para qué nace Git?

02

Por qué usar Git?

Beneficios de este VCS.

03

Cómo funciona Git?

Los tres estados de Git.

Flujo de trabajo,

Gitignore, Set mínimo de archivos, Comando básicos.

04

Trunk Based Development

¿Qué es Trunk Based Development?

05

GitLab

¿Qué es GitLab? Issue, Milestone, Merge Request.

06

¿Cómo trabajamos en equipo?

Flujo de trabajo

07

Demo

Breve historia



Breve historia

Antes que nada, **qué es un control de versiones?**

Es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo.

Hoy en día hay varias firmas, como por ejemplo:



| Entonces, cómo y para qué nace Git?

Git nace en 2005 por la ruptura entre la comunidad que desarrollaba el núcleo de Linux y la compañía que desarrollaba BitKeeper, un DVCS que dejó de ser gratuito. Debido a esto, la comunidad de Linux, en particular Linus Torvalds, impulsó su propia herramienta basada en el uso de BitKeeper.

| Entonces, cómo y para qué nace Git?

Sus objetivos fueron:

- Velocidad
- Diseño sencillo
- Fuerte apoyo al desarrollo no lineal
- Completamente distribuido
- Capaz de manejar grandes proyectos de manera eficiente

Por qué Git?



| Integridad

Todo en Git es verificado mediante un checksum antes de ser almacenado.

Esta funcionalidad está integrada en Git al más bajo nivel y es parte integral de su filosofía, **no se puede perder información o sufrir corrupción de archivos sin que Git lo detecte.**

| Integridad

El mecanismo que usa Git para generar esta suma se conoce como **SHA-1** y se visualiza de la siguiente manera:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

| Velocidad

Git sólo necesita archivos y recursos locales para operar, a diferencia de un VCS donde la mayoría de las operaciones tienen sobrecarga del retardo de la red.

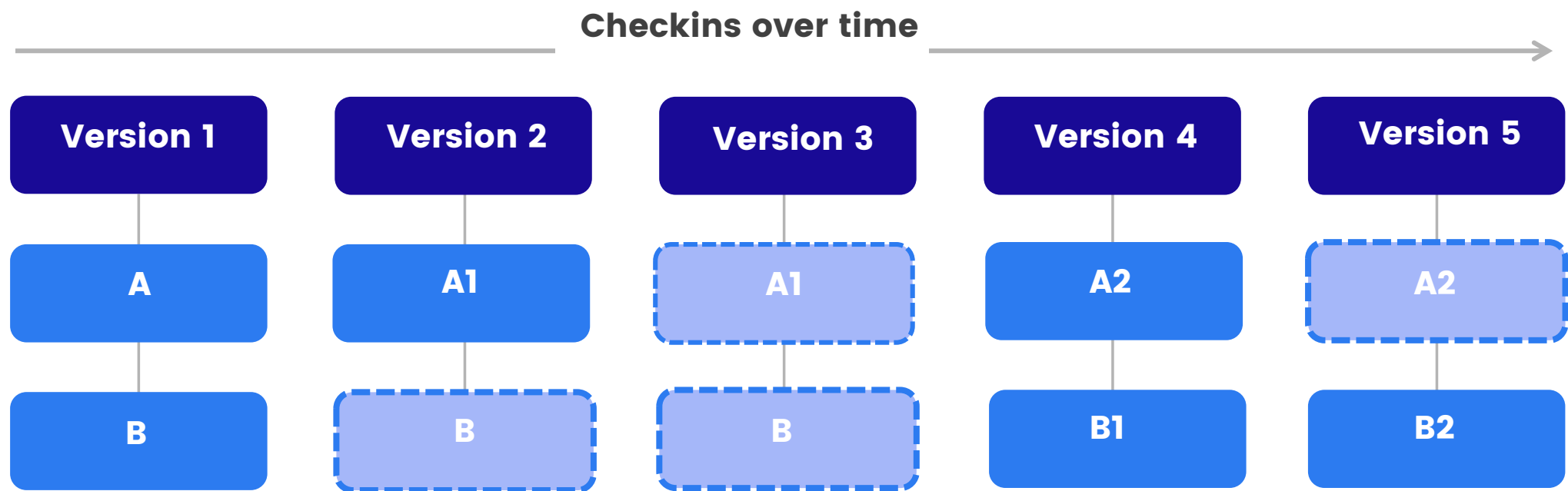
Al tener toda la historia del proyecto en tu disco local, la mayoría de las **operaciones** son prácticamente **inmediatas**.

| Eficiencia

Cada vez que confirmamos un cambio, Git "hace una foto" del aspecto de todos los archivos y genera una referencia a esa instantánea.

Para sumar eficiencia, **si no realizamos cambios en un archivo, Git no lo almacena de nuevo.**

Eficiencia



Cómo funciona Git?

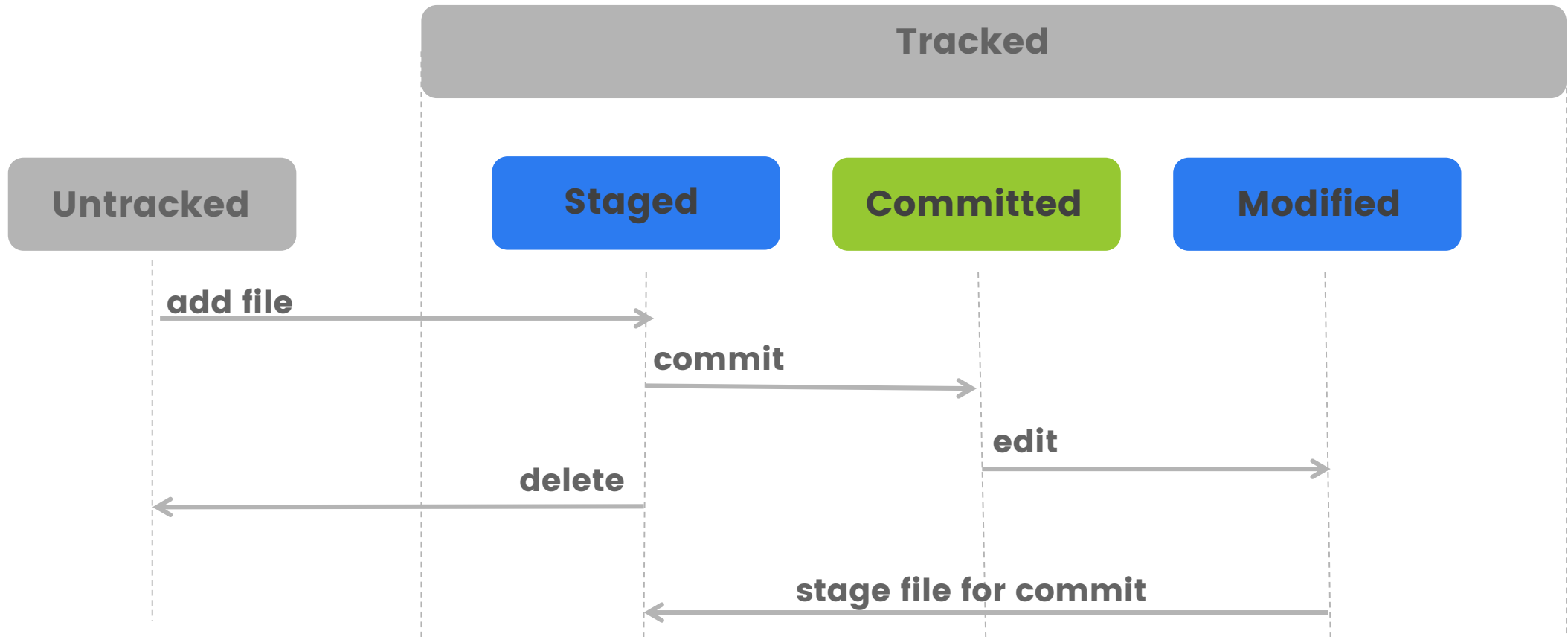


Tres estados

Git tiene tres estados en los que se pueden encontrar tus archivos:

- **Modified** (el archivo se modificó pero todavía no se confirmó en tu base de datos).
- **Staged** (el archivo está modificado en su versión actual y listo para subirse).
- **Committed** (los datos están almacenados de manera segura en tu base de datos local).

Tres estados



| Flujo de trabajo

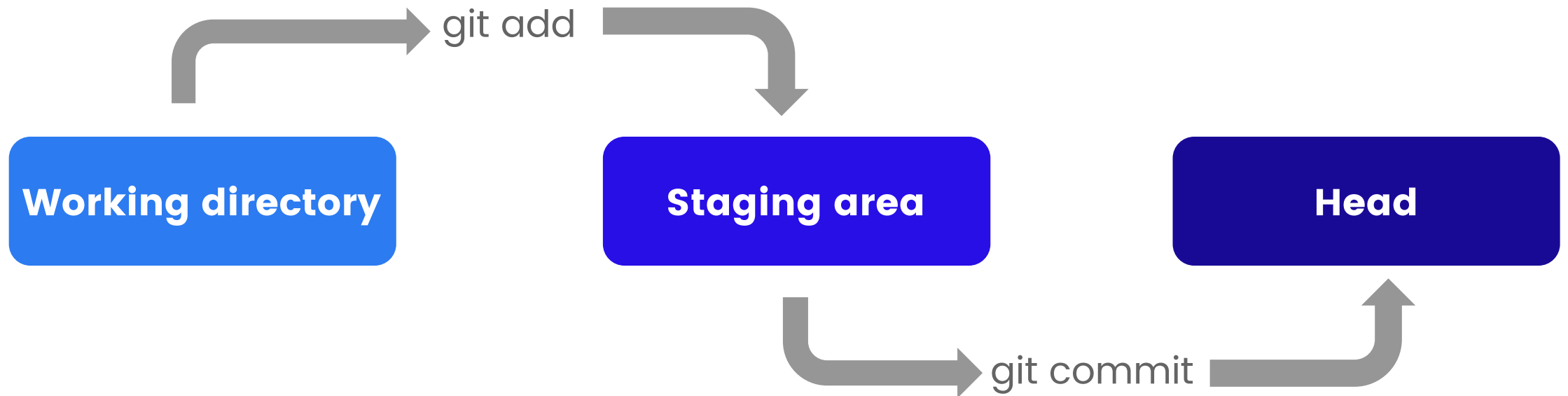
El repositorio local está compuesto por tres “árboles” administrados por Git.

Working directory que contiene los archivos de manera local.

Index o **staging área** que básicamente actúa como un intermediario.

Head o git directory (repository) que apunta al último commit realizado y es donde Git almacena los metadatos para el proyecto.

Flujo de trabajo



Gitignore

Git tiene un archivo interno donde en el cual se puede especificar que tipos de **archivo van a ser ignorados**, es opcional y puede o no estar dentro del repositorio.

Ejemplo:

```
.gitignore 5.73 KB
1  ## Ignore Visual Studio temporary files, build results, and
2  ## files generated by popular Visual Studio add-ons.
3  ##
4  ## Get latest from https://github.com/github/gitignore/blob/master/VisualStudio.gitignore
5
6  # User-specific files
7  *.rsuser
8  *.suo
9  *.user
10 *.userosscache
11 *.sln.docstates
12
13 # User-specific files (MonoDevelop/Xamarin Studio)
14 *.userprefs
15
16 # Build results
17 [Dd]ebug/
18 [Dd]ebugPublic/
19 [Rr]elease/
20 [Rr]eleases/
21 x64/
22 x86/
```

Set mínimo de archivos

Por cada repositorio que se crea, tenemos ciertos **archivos requeridos**:

- .gitignore
- Readme.md
- Changelog.md
- License
- .gitlab-ci.yml

Comandos básicos

- `git help`: muestra una lista con comandos más utilizados en Git
- `git init`: crea un repositorio de git
- `git add + path`: agrega al repositorio los archivos que indiquemos
- `git add -A`: agrega al repositorio TODOS los archivos y carpetas que estén en nuestro proyecto
- `git commit -m "mensaje" + archivos`: hace commit a los archivos que indiquemos, quedando guardadas las modificaciones
- `git commit -am "mensaje"`: hace un commit de los archivos que hayan sido modificados (después de hacer un `git add`)

Comandos básicos

- `git branch`: muestra una lista de los branches que existen en nuestro repositorio
- `git checkout NombreDelBranch`: sirve para moverse entre branches
- `git merge NombreDelBranch`: hace un merge entre dos branches, en este caso la dirección del merge sería entre el branch que indiquemos en el comando y el branch donde estemos ubicados.
- `git status`: nos indica el estado del repositorio, qué archivos están modificados, cuáles no están siendo seguidos por Git, etc.
- `git clone URL/name.git NombreProyecto`: clona un proyecto en la carpeta NombreProyecto

Comandos básicos

- `git checkout -b NombreDelBranch`: crea un nuevo branch y se cambia automáticamente a él, clonando el branch desde donde ejecutamos el comando.
- `git push origin NombreDelBranch`: luego de un `git commit`, con este comando podemos subir los archivos al repositorio remoto, específicamente al branch que indiquemos
- `git pull origin NombreDelBranch`: actualiza nuestro branch local con los cambios que tenga el branch remoto que indiquemos en el comando

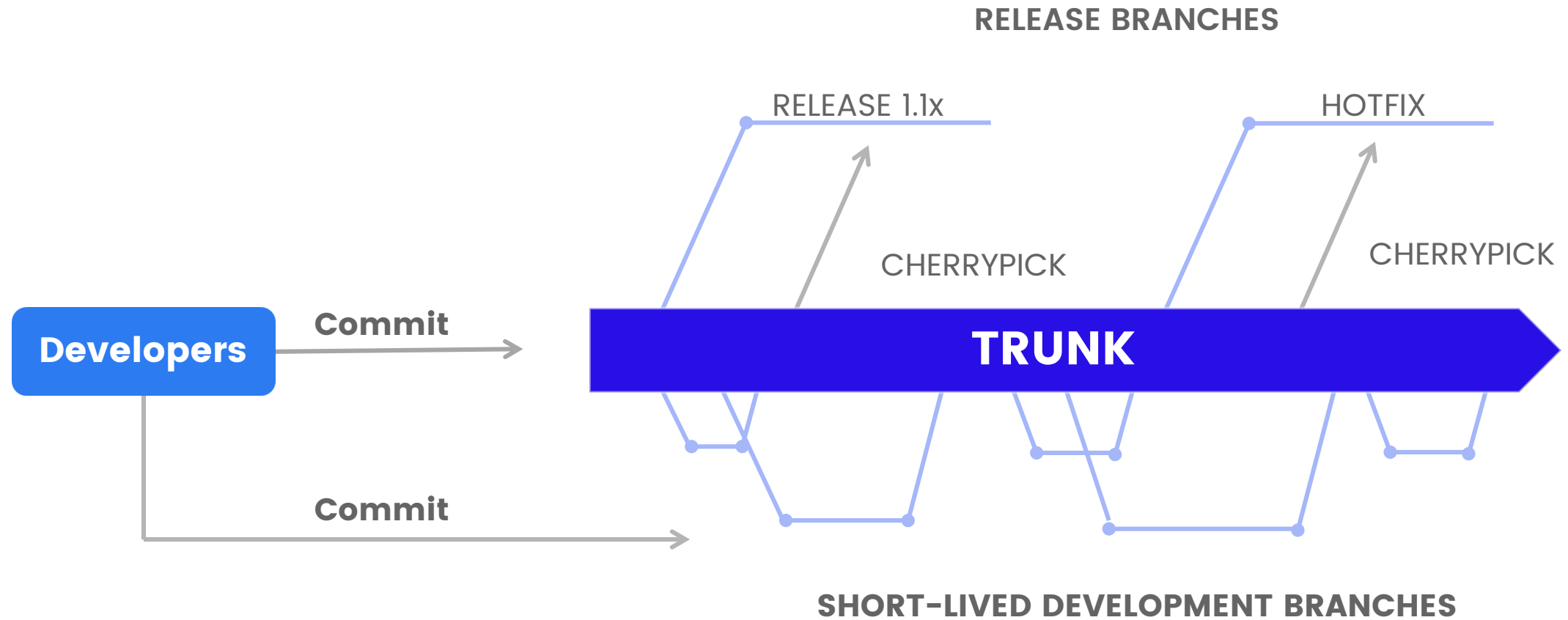
Trunk Based Development



| Trunk Based Development

Basado en **branches** de corta duración, **code review** de forma fluida y compilaciones automatizadas en **CI**, garantiza que la base de código siempre sea limpia y ayuda a la **entrega continua**.

CI - Trunk Based Development



GitLab



| Qué es GitLab?

Es un servicio que nos va a permitir mantener centralizados nuestros **repositorios de código** y el **orden** en ellos.

Dentro de GitLab vamos a encontrar **Issues**, **Milestones** y **Merge request**.



| Issues

Es la **unidad mínima de trabajo** dentro de GitLab, vamos a poder clasificarlas con **labels** para clasificar su ciclo de vida y mantenerlas ordenadas en **boards**.

Issues

Active projects > La Nación > Bonvivi > Issues

Open 62 Closed 158 All 220

    Edit issues [New issue](#)



Search or filter results...

Created date



Maquetado en componentes (BVA-203)




#220 · opened 3 hours ago by Tomas Wainberg

Sprint 5

Back Office

Doing

Task

  1  0

updated 3 hours ago

prefiero que me llamen

#219 · opened 4 hours ago by Vanesa Graciela Germade

Enhancement

 0

updated 4 hours ago

Milestones

Normalmente usamos los Milestones como un **espacio de trabajo** donde se realiza el sprint, seteando una **fecha de inicio y otra de fin**.

Si trabajamos ordenadamente, al Milestone le asociamos Issues y Merge Request para **trackear el trabajo**.

Milestones

Active projects > La Nación > Bonviviir > Milestones > Sprint 5

Past due Milestone Mar 25, 2019–Apr 5, 2019

Edit

Promote

Close milestone

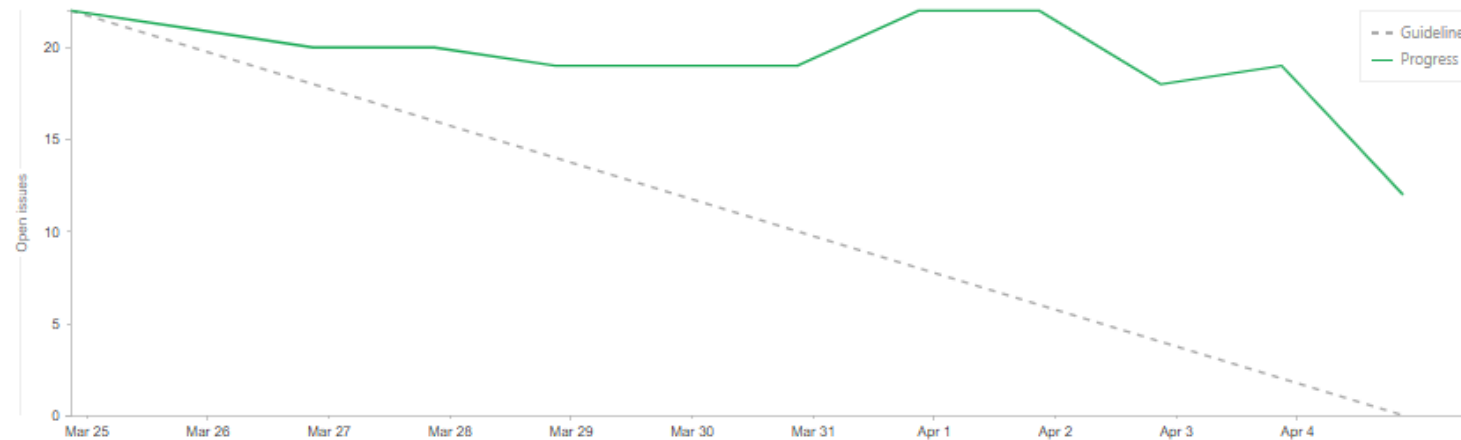
Delete

Sprint 5

Burndown chart

Issues

Issue weight



The tabs below will be removed in a future version

Learn more about [issue boards](#), to keep track of issues in multiple lists, using labels, assignees, and milestones. If you're missing something from issue boards, please create an issue on [GitLab's issue tracker](#).

Issues 41

Merge Requests 26

Participants 5

Labels 14


Merge Request

En cuanto tomamos un issue, generamos un merge request para llevar el tracking del proyecto. GitLab también nos permite generar **code reviews** gracias a esta práctica.

Merge Request

Denise Krzyny > Test > Merge Requests > 38

Open

Opened 1 minute ago by  Denise Krzyny

Edit

Close merge request

WIP: Resolve "Replace list items text."

Closes #6



Request to merge 6-replace-list-items-... into master

Open in Web IDE

Check out branch




No approval required



Merge

This is a Work in Progress

 Resolve WIP status

Closes #6

[Assign yourself to this issue](#)

You can merge this merge request manually using the [command line](#)



0



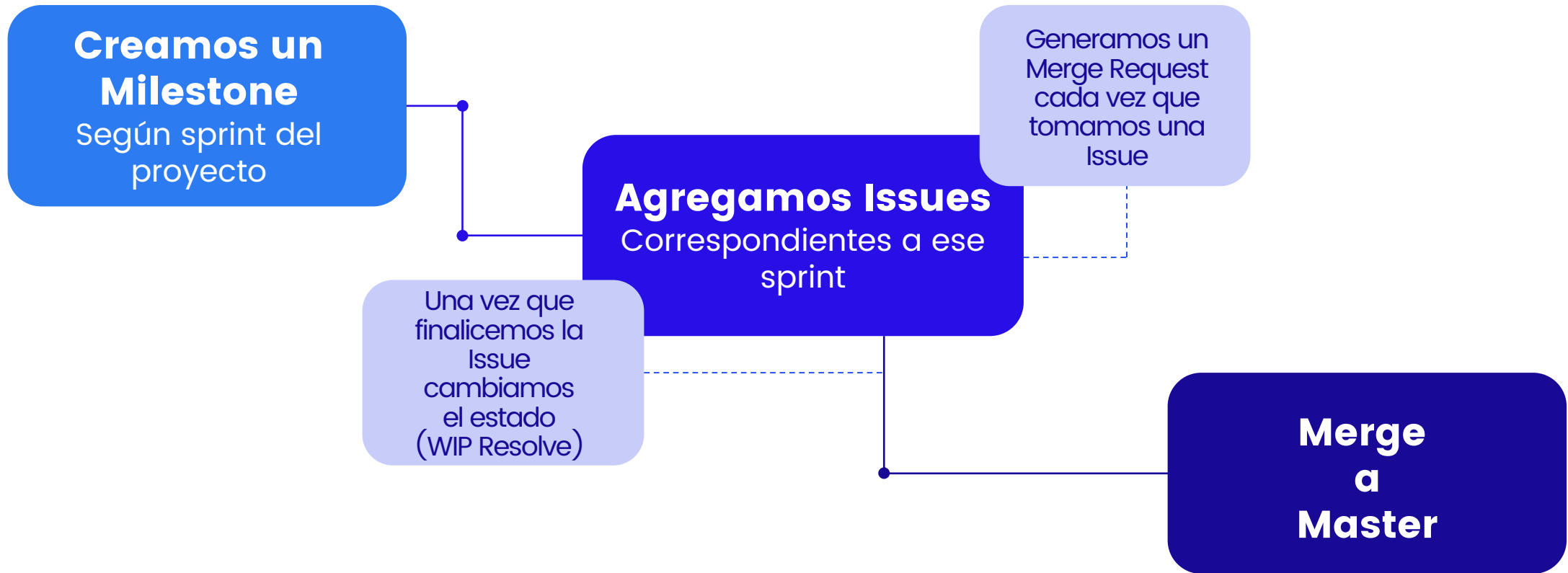
0



**Cómo trabajamos en
equipo?**



Flujo de trabajo



Demo





Let's keep in touch

Argentina | Buenos Aires
+54 (11) 4982 4185
info@lagash.com

Chile | Santiago de Chile
+56 (2) 2231 9428
info_chile@lagash.com

Colombia | Bogotá
+57 (1) 750 5276
info_colombia@lagash.com

México | México DF
+52 (55) 6394 0617
info_mexico@lagash.com

USA | Seattle - US
+1 844 4 Lagash
infousa@lagash.com

Uruguay | Montevideo
+598 2623 2546
info_uy@lagash.com